PRELIMINARY
DAP ASSEMBLY PROGRAM
FOR THE DDP-24
GENERAL PURPOSE COMPUTER

COMPUTER CONTROL COMPANY

CONTENTS

SECTION I

INTRODUCTION

The DDP-24 Assembly Program (DAP) is a programming aid that will translate a symbolic language to permit writing programs in a form more convenient to the programmer while maintaining the flexibility of machine language (binary) coding. It enables the substitution of mnemonic symbols for desired binary instructions, such as ADD in place of 001000, and allows the programmer to assign names to specific data items or groups such as DAY, RATE, or MACH and to use these names when referring to the items as operands. Several pseudo-operations are provided to allow the programmer to express concepts that have no counterpart in normal machine language. Additional features of DAP include the facility for programmer defined macro-operations, the capability for linking with FORTRAN II programs and vice versa, the option of producing absolute or relocatable object programs, and operations that give the flexibility of using subroutines (either library or non-library).

DAP is designed to work with a minimum machine configuration: a paper tape reader, a paper tape punch, an on-line typewriter and 4,096 words of memory. No optional features are required to assemble a program using DAP; however, provision is made for utilization of a line printer, magnetic tapes and card equipment when available. Thus, by using a smaller system the user may assemble a program designed to run on, and take advantage of a more complex system.

SECTION II

SOURCE LANGUAGE FORMAT

GENERAL

The general format of the DAP source language consists of four major subsections; these are the Location, Operation, Variable and Comments Fields. The rules governing the use of these fields are the same for either a paper tape system or a card system; however, the preparation of the source program for processing by DAP (keypunching) is different for the two systems. This difference is discussed under SOURCE PROGRAM PREPARATION.

THE CODING FORM

Figure 1 illustrates the coding form that may be used for writing DAP source programs for either the paper tape system or the card system. The line and column spacing on the form has been designed to be compatible with standard typewriter spacing and the color (dark green print on light green paper) has been used to minimize reflective glare.

THE LOCATION FIELD

The Location Field is normally used to assign a symbolic label to an instruction, constant, or buffer area when it is necessary to refer to that location elsewhere in the program. The symbolic label in the Location Field consists of from one to four characters from among the thirty-six character set composed of the alphabet and the ten numeric digits. At least one of the characters in any label must be alphabetic.

Labels provide the means of symbolic addressing in programs. When a label occurs in the Location Field of the input format, it is assigned the current value of the DAP location counter (unless an EQU or ORG operation in the Operation Field causes it to be assigned otherwise). The first such occurrence constitutes the definition of the label and any subsequent occurrence will cause redefinition and an error print-out.

THE OPERATION CODE FIELD

Mnemonic operation codes are used to represent machine instructions, commands to the assembler itself, and macro-operations. Operation codes are either 3 or 4 characters in length. Appendix B summarizes operation codes recognized by DAP. In addition to specifying an operation, the Operation Field is also used to specify whether indirect addressing is desired. This is indicated by writing an asterisk (*) immediately following the operation code.

PROGRAMMER					DATE	PAGE OF
PROGRAM						CHARGE
LOCATION	ⓣ	OPERATION	ⓣ	ADDRESS, X	ⓣ COMMENTS	ⓒR IDENTIFICATION
1 4		6 10		12	30 72	73 80

Figure 1. DAP Coding Form

THE VARIABLE FIELD

The Variable Field has different uses for different classes of commands. The specific use for a particular command is covered under the explanation of that command. The most general use of this field is for the specification of an address and index register. One may write any permissible expression (an expression is defined below) to represent the address portion of the command. If this expression is not followed by a comma, no index register is specified. An index register is specified by following the address expression with a comma and another expression indicating the desired index register.

THE COMMENTS FIELD

This field may be used for any comments the programmer cares to write. The Comments Field has no effect on the assembler, but is printed out on the symbolic listing during Pass Two, if a listing has been requested.

SYMBOLS AND EXPRESSIONS

SYMBOLS

Symbols consist of one to four characters, one of which must be alphabetic. The remaining characters, if any, may be any alphabetic and/or numeric combination. The following 11 characters may not be used as part of a symbol:

+ (plus sign)	' (apostrophe)
- (minus sign)	, (comma)
* (asterisk)	((left parenthesis)
/ (slash)	) (right parenthesis)
$ (dollar sign)	& (ampersand)
= (equal sign)	

The period (.) may be used as a character in a symbol, but the programmer is cautioned to avoid possible confusion with the decimal literal.

EXPRESSIONS

An expression may be either simple (composed of a single element) or compound (composed of two or more elements separated by operators). Both expression types may have either "relocatable" or "absolute" modes. A relocatable expression is one which is relative to the first instruction of the program; an absolute expression is one which has a constant value regardless of its relative position in the program. The overall mode of the expression depends on the mode of each of the individual elements used to make up the expression.

An element is the smallest component of an expression. An element is either a symbol, a decimal integer less than 2^{23}, or an octal integer less than 2^{23}. An octal integer is denoted by a preceding apostrophe (e.g., '123). (An asterisk may also be used as an element as described below.)

An operator may be used to separate elements in compound expressions. The operators have the following meanings:

plus or ampersand	addition
minus	subtraction
asterisk	multiplication
slash	division

An asterisk may also be used as an element, as explained under ASTERISK CONVENTIONS.

The multiplication and division operators take precedence over the addition and subtraction operators. Parentheses are not allowed; therefore, implied parenthetical groupings are processed from left to right, for example:

A*B+C*D	is interpreted as (A*B) + (C*D)
A+B*C	is interpreted as A + (B*C)
A-B+C	is interpreted as (A-B) + C
A*B/C	is interpreted as (A*B)/C
A/B/C	is interpreted as (A/B)/C

Two operators may not appear in succession (e.g., A++B); however, when the asterisk is used as an element, its meaning is unambiguous and is therefore allowed. For example,

```
*+1
```

means "this location plus one",

and

```
**2
```

means "this location times two".

LITERALS

It is often necessary to refer to a memory location containing a constant to be defined by the programmer. This can easily be done by the use of one of the data defining pseudo-operations provided in the DAP language. However, it is sometimes more convenient to reference a constant literally rather than symbolically. Consider the following examples:

```
        .
        .
        .
       LDA    ONE
        .
        .
        .
ONE    DEC    1
       END
        .
        .
        .
       LDA  =  1
        .
        .
        .
       END
```

In the first example the decimal constant 1 is referred to symbolically as ONE and must be defined by the programmer; however, in the second example, the programmer is not required to define the constant. DAP will interpret "=1" as a literal, will automatically assign a location for the constant at the end of the program, and will insert the address of that location in the LDA instruction.

Three different types of literals are interpreted by DAP: decimal, octal and alphanumeric.

1) Decimal Literals.

A decimal literal consists of the equals character (=) followed by a signed or unsigned fixed-point decimal number (Decimal Numbers are discussed in Section III).

2) Octal Literals

An octal literal consists of the equals character (=) followed by an apostrophe (') followed by a signed or unsigned octal number (refer to the discussion of Octal Numbers).

3) Alphanumeric Literals.

An alphanumeric literal consists of the equals character (=) followed by the letter A followed by four alphanumeric characters. With the exception of the space, control characters cannot be used in an alphanumeric literal (e.g., carriage return, tab, stop code, etc).

ASTERISK CONVENTIONS

The following programming conventions using the asterisk are allowed by DAP:

1) * in column 1 or first element in Location Field; treat entire card or line as remarks.

2) * appended to instruction mnemonic: set indirect address flag.

3) * an element: current value of the Location Counter.

4) ** as a symbolic address: address will be modified by another instruction.

5) *** as an operation code: op-code will be modified by another instruction.

SOURCE PROGRAM PREPARATION

PAPER TAPE

In order to make more efficient use of the paper tape as an input medium to DAP, a terminating code has been used to define the difference between fields rather than specifying "columns." For example, if the Location Field is not used, it is not necessary to space five times in order to be in position for the Operation Field. The terminating code used is the tab. In addition, the carriage return will terminate the entire line. In the example used above, a tab would immediately define the start of the Operation Field. The general format for the entire line would be:

Location Field (tab) Operation Field (tab) Variable Field (tab) Comments Field (carriage return)

CARDS

When using cards, no purpose is served by trying to make a line of code more compact since the entire card must be read. Therefore, the card columns are used to define the fields. The only exception to this is the termination of the Variable Field and the start of the Comments Field. DAP

will assume the Comments Field to start after the first blank column following the Variable Field. If a blank is embedded within the Variable Field, DAP will assume the remainder of the line to be comments. The general format for the card would be:

Location Field	Columns 1 to 4
Operation Field	Columns 6 to 10
Variable Field	Columns 12 to first blank column
Comments Field	First blank column to column 72
Identification Field	Columns 73 to 80

SECTION III

PSEUDO-OPERATIONS

GENERAL

In addition to translating all of the DDP-24 instruction mnemonics, DAP will also translate certain pseudo-operations specifying optional controls and programmer aids for number conversions. Operations that fall into this category are called "pseudo" because they have no counterpart in the list of actual DDP-24 instructions.

Primarily, pseudo-operations are provided to give the programmer a flexible language. It is possible to generate equivalent information by using different instructions or pseudo-operations; however, the choice made by the programmer is often intended to be meaningful in the context of the program listing for the convenience of others who may examine the program. For example, the octal word 40000144 could be generated by any one of the following DAP operations:

```
HLT*    '144
HLT*     100
MZE      100
PZE*    '144
BCI      1,-01M
DEC     -100
OCT     -144
```

ASSEMBLY CONTROLLING PSEUDO-OPERATIONS

The pseudo-operations in this category (ABS, END, MOR and REL) are used for directing DAP to perform various assembly functions; they do not generate instructions in the object program.

ABS

The ABS (ABSOLUTE) pseudo-operation is used to direct DAP to assemble the subsequent instructions in the absolute mode. The format for using the ABS pseudo-operation is:

LOCATION	Ignored
OPERATION	ABS
VARIABLE	Ignored
COMMENTS	Normal

The effect of the ABS pseudo-operation is to cause a termination of the current block of output information and the start of a new block in which words are assigned absolute locations. The assembler will then continue to run in the absolute mode until a REL or ORG pseudo-operation is encountered. Initialization of the assembler automatically sets the absolute mode.

END

The END (END) pseudo-operation is used to direct DAP to terminate the current assembly pass and prepare for the next pass. The format for using the END pseudo-operation is:

LOCATION	Ignored
OPERATION	END
VARIABLE	1) Main Program. An expression that defines the address of the instruction to which control should be transferred at the conclusion of the loading process at object time. 2) Subroutine. Ignored.
COMMENTS	Normal

The END pseudo-operation causes DAP to perform the following functions:

1) If in pass one, halt. When the start button is depressed, start processing pass two (while the computer is halted, the operator must reposition the source tape to the beginning).

2) If in pass two:

a) The current block of assembly output information is terminated.

b) The transfer vector is tested to see if subroutines are required; if so, a request is typed for the subroutine tape and a pass is made to fetch the requested subroutines.

c) If this is a main program, a jump record is written following the assembly output. The address of the jump is the value of the expression in the Variable Field. If this is a subroutine, no jump record is written.

d) The assembly process is terminated.

If an END pseudo-operation is used, it must be the last operation of the source program.

MOR

The MOR (MORE) pseudo-operation has various meanings which depend on Sense Switch settings (see below). The format for using the MOR pseudo-operation is:

LOCATION	Ignored
OPERATION	MOR
VARIABLE	If the MOR pseudo-operation is to be treated as an END pseudo-operation, the rules for the Variable Field are the same as those described for the END pseudo-operation; otherwise this field is ignored.
COMMENTS	Normal

The MOR pseudo-operation causes DAP to perform the following functions:

1) If Sense Switch 4 is down, halt. When the start button is depressed, interrogate Sense Switch 5 (this will allow the operator to change paper tapes in the event the source program has been produced in more than one piece).

2) If Sense Switch 4 is up, do not halt, but interrogate Sense Switch 5.

3) If Sense Switch 5 is down, continue the assembly process.

4) If Sense Switch 5 is up, treat the MOR pseudo-operation as an END pseudo-operation.

ORG

The ORG (ORIGIN) pseudo-operation is used to assign a new value to the Location Counter. The format for using the ORG pseudo-operation is:

LOCATION	Normal
OPERATION	ORG
VARIABLE	Any relocatable or absolute expression (if left blank, an absolute origin of zero will be assumed). Any symbol used in this field must have been previously defined.
COMMENTS	Normal

The ORG pseudo-operation performs the following functions:

1) The expression in the Variable field is evaluated for value and mode (relocatable or absolute).

2) The location counter is reset to the value thus determined.

3) If there is a symbol in the Location Field, it is given this value and mode.

The ORG pseudo-operation will cause DAP to switch to relocatable or absolute mode and will continue in this mode until an ABS, REL or another ORG is encountered.

REL

The REL (RELOCATABLE) pseudo-operation is used to direct DAP to assemble the subsequent instructions in the relocatable mode. The format for using the REL pseudo-operation is:

LOCATION	Ignored
OPERATION	REL
VARIABLE	Ignored
COMMENTS	Normal

The effect of the REL pseudo-operation is to cause a termination of the current block of output information and the start of a new block in which words are assigned relative locations. The assembler will then continue to run in the relocatable mode until an ABS or ORG pseudo-operation is encountered.

DATA DEFINING PSEUDO-OPERATIONS

The pseudo-operations in this category (BCI, DEC and OCT) are used for the generation of data to be included as part of the object program. The following is a discussion of the three different types of data that can be interpreted by DAP.

1) Alphanumeric Data.

Certain restrictions are placed on the use of alphanumeric data within the scope of the data defining features of DAP (for both the BCI pseudo-operation and the alpha-numeric literal). These restrictions have to do with the control characters: back space, lower case shift, upper case shift and carriage return. With the exception of these specific characters, all of the characters listed in Appendix E may be used in alphanumeric data fields.

2) Decimal Data.

a) Fixed-point decimal data.

A significance of six decimal digits can be maintained in single precision fixed-point arithmetic on the DDP-24. In many arithmetic operations, this degree of significance is adequate and is desirable because of the speed in computation. DAP provides two facilities for generating fixed-point numbers using decimal notation: decimal literal and the DEC pseudo-operation.

A fixed-point decimal number requires one computer word (sign and 23 bits of significance) and is specified in the DAP language by the use of two parts:

i) The significant part, which is a signed or unsigned decimal number with or without a decimal point. If the decimal point is not specified, it is assumed to be immediately to the right of the last digit (a decimal integer).

ii) The scaling part, which is the letter B followed by a signed or unsigned decimal integer specifying the position of the understood binary point. The scaling part need not be present (in which case the number will be a truncated decimal integer whose understood binary point is immediately to the right of the least significant bit in the computer word -- position 23).

The general form of the scaling part is B±nn, where nn gives the position of the understood binary point relative to the machine binary point, the - defines the understood binary point to be to the left of the machine binary point, and the + (or no sign) defines the understood binary point to be to the right of the machine binary point. The machine binary point is defined to be between the sign bit and the most significant bit of the computer word (between positions 1 and 2).

The following are examples of how DAP would produce fixed-point numbers. The right column shows the decimal number to be translated and the left column shows the resulting octal word that would be generated by DAP.

00000017^	15
00000017^	+15.14
00000017^	15B+23
17^000406	15.001B5

^ indicates understood binary point

Fixed-point numbers are limited to a magnitude less than 2^{23}.

b) Floating-point decimal data.

There are two types of floating-point numbers available to the DAP user, both of which require two computer words; these are 1) single precision floating-point (sign and 23 bits of mantissa, sign and 8 bits of exponent), and 2) extended precision floating point (sign and 38 bits of mantissa, sign and 8 bits of exponent). Figure 2 shows the formats of floating point numbers. A floating point decimal number may be defined by the use of the DEC pseudo-operation only. A floating point decimal number is specified in the DAP language by the use of two parts:

i) The mantissa part, which is a signed or unsigned decimal number preceded by a decimal point.

ii) The exponent part, which is one letter E or two E's followed by a signed or unsigned decimal integer specifying the power of ten which is the coefficient of the mantissa. The single letter E specifies a single precision floating-point number; a double letter E specifies an extended precision floating-point number. An exponent part must be specified for floating-point numbers; if omitted, DAP will assume the mantissa to be a fixed-point decimal integer. The magnitude of the exponent may not exceed 75 decimal (coefficient = 10^{75}).

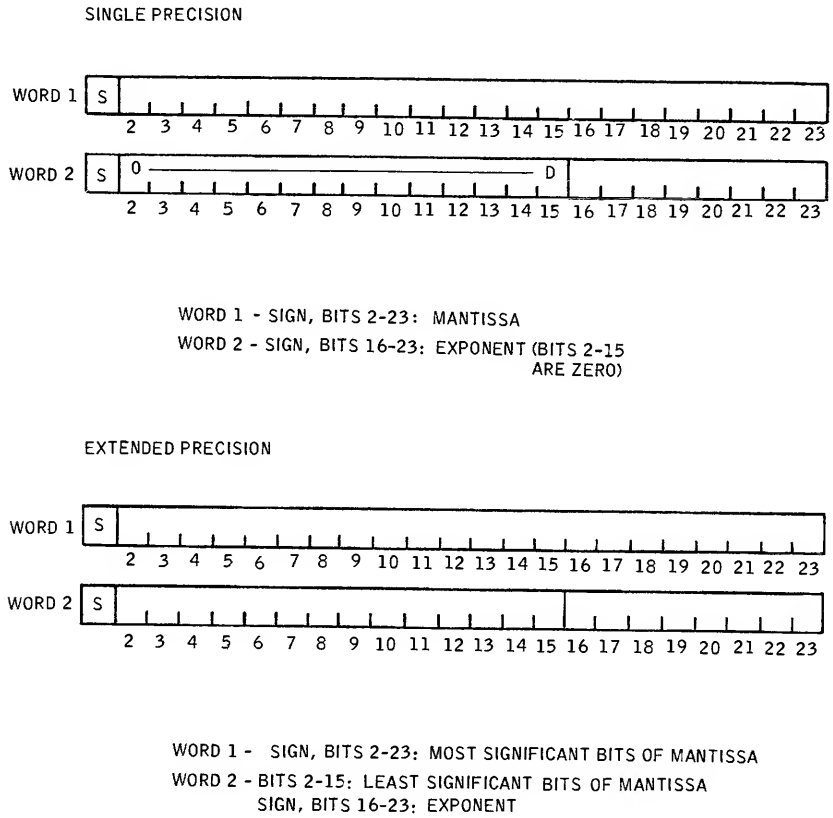


Figure 2. Floating Point Format

The following are examples of how DAP would produce floating-point numbers. The right column shows the decimal number to be translated and the left column shows the resulting two octal words that would be generated by DAP (assigned to successive storage locations). The second octal word for each example contains the exponent (power of two) in the low order 2-2/3 octal digits; the sign bit of the second octal word is the sign of the exponent. All numbers are normalized.

```
36000000
00000004          .15E2

76000000
00000004          -.15E2

36000000
00000004          +.15E+2

30000000
00000001          .15E1

31463146
54631403          .1EE0

71463146
54631403          -.1EE+00
```

iii) Octal data.

DAP provides two facilities for generating octal numbers using octal notation: octal literal and the OCT pseudo-operation. The only allowable characters in an octal data field are: + - 0 1 2 3 4 5 6 7

Octal numbers may be signed or unsigned and are limited to a magnitude that is less than 2^{23}.

BCI

The BCI (BINARY CODED INFORMATION) pseudo-operation is used to direct DAP to generate binary words from alphanumeric data. The format for using the BCI pseudo-operation is:

LOCATION	Normal
OPERATION	BCI
VARIABLE	n, followed by 4n alphanumeric characters. n specifies the number of words to be converted and may not exceed 10 decimal.
COMMENTS	Normal

The effect of the BCI pseudo-operation is to convert each group of four characters into a binary word; these words are stored in successively higher storage locations as the variable field is processed from left to right. If there is a symbol in the location field, it refers to the first word of data generated.

DEC

The DEC (DECIMAL) pseudo-operation is used to direct DAP to generate binary words from decimal data. The format for using the DEC pseudo-operation is:

LOCATION	Normal
OPERATION	DEC

VARIABLE	One or more subfields, each containing a decimal data item. The subfields are separated by commas; the number of subfields permissible is limited only by the restriction that the last subfield must be terminated by a tab code or carriage return.
COMMENTS	Normal

The effect of the DEC pseudo-operation is to cause DAP to convert each subfield to one or two binary words depending on the decimal data being fixed-point or floating-point, respectively. These words are stored in successively higher storage locations as the variable field is processed from left to right. If there is a symbol in the Location Field, it refers to the first word of data generated.

OCT

The OCT (OCTAL) pseudo-operation is used to direct DAP to generate binary words from octal data. The format for using the OCT pseudo-operation is:

LOCATION	Normal
OPERATION	OCT
VARIABLE	One or more subfields, each containing an octal data item. The subfields are separated by commas; the number of subfields permissible is limited only by the restriction that the last subfield must be terminated by a tab code or carriage return.
COMMENTS	Normal

The effect of the OCT pseudo-operation is to cause DAP to convert each subfield to a binary word; these words are assigned to successively higher storage locations as the Variable Field is processed from left to right. If there is a symbol in the Location Field, it refers to the first word of data generated.

DICTIONARY CONTROLLING PSEUDO-OPERATIONS

The pseudo-operations in this category (CALL, NTRY and RTRN) are used for the generation of subroutine linkage facilities to allow communication between programs.

CALL

The CALL (CALL) pseudo-operation is used to direct DAP to generate instructions that will transfer control to a specified subroutine. The format for using the CALL pseudo-operation is:

LOCATION	Normal
OPERATION	CALL
VARIABLE	A subroutine name.
COMMENTS	Normal

The effects of the CALL pseudo-operation are:

1) To enter the subroutine name from the Variable Field into the transfer vector if it is not already there.

2) To enter into the sequence of assembled instructions a JST* with an address that is the transfer vector location containing the Variable Field subroutine name.

3) If there is a symbol in the Location Field, it is assigned to the JST* instruction inserted in step 2 above.

NTRY

The NTRY (ENTRY) pseudo-operation is used to define a DAP subroutine and to symbolically assign a name to the subroutine for external reference. The format for using the NTRY pseudo-operation is:

LOCATION	The name of the subroutine
OPERATION	NTRY
VARIABLE	A name defining the entry point of the subroutine (if left blank, the first executable instruction of the subroutine will be assumed to be the entry point).
COMMENTS	Normal

The effect of the NTRY pseudo-operation is to cause the name in the Location Field to be punched on the paper tape output as identification for the subroutine library (refer to Section IV). There may be as many NTRY pseudo-operations in a subroutine as there are entry points; however, the NTRY pseudo-operation must be the first operation of the subroutine, preceded only by another NTRY, if present.

RTRN

The RTRN (RETURN) pseudo-operation is used to direct DAP to generate an instruction that will transfer control back to a calling program. The format for using the RTRN pseudo-operation is:

LOCATION	Normal
OPERATION	RTRN
VARIABLE	Ignored
COMMENTS	Normal

The effect of the RTRN pseudo-operation is to cause a jump instruction to be generated with the appropriate address specified by the use of the NTRY pseudo-operation. There may be more than one RTRN in a subroutine and may be placed anywhere within the subroutine as an exit from the routine.

LIST CONTROLLING PSEUDO-OPERATIONS

The pseudo-operations in this category (LIST and NLST) are used to specify the listing options provided in DAP.

LIST

The LIST (LISTING) pseudo-operation is used to direct DAP to produce a side-by-side listing of the program being assembled. The format for using the LIST pseudo-operation is:

LOCATION	Ignored
OPERATION	LIST
VARIABLE	Ignored
COMMENTS	Normal

The effect of the LIST pseudo-operation is to cause the source program and its octal representation to be listed on the on-line typewriter. The assembler then continues to operate in the "listing" mode until an NLST pseudo-operation is encountered.

NLST

The NLST (NO LISTING) pseudo-operation is used to direct DAP to refrain from producing a side-by-side listing of the program being assembled. The format for using the NLST pseudo-operation is:

LOCATION	Ignored
OPERATION	NLST
VARIABLE	Ignored
COMMMENTS	Normal

The effect of the NLST pseudo-operation is to cause DAP not to produce a listing of the source program and its octal representation on the on-line typewriter. The assembler than continues to operate in the "no-listing" mode until a LIST pseudo-operation is encountered (initialization of the assembler automatically sets the "no listing" mode).

MACRO DEFINING PSEUDO-OPERATIONS

A macro operation is defined by the use of the MAC and ENDM pseudo-operations. MAC defines the start of the macro operation, identifies the operation by a unique name, and supplies a list of parameters for which symbols may be substituted each time the macro-operation is used. ENDM terminates the definition of the macro-operation. After a macro-operation has been defined, it may be used as often as desired. A macro sequence may be the following:

```
MACA  MAC
      LDA   X
      ADD   Y
      STA   Z
      ENDM
```

Each time the pseudo-operation, MACA, appears in an instruction sequence, it is replaced by the three instructions defined by the above macro. For example, the coded sequence:

```
      FMB   X + 2, 1          READ TWO WORDS
      MACA
      OTM   Z                 OUTPUT SUM
```

would be assembled as:

```
      FMB   X + 2, 1          READ TWO WORDS
      LDA   X
      ADD   Y
      STA   Z
      OTM   Z                 OUTPUT SUM
```

Thus, it is seen that the macro-instruction MACA can be regarded as an abbreviation for a sequence of instructions.

In most instances, it is undesirable to have the repetitive sequence of instructions operate on exactly the same data fields in exactly the same manner each time the macro substitution is used. It would be more convenient if the same general pattern could be repeated, but with certain substitutions depending on the requirements. This type of substitution is possible in the DAP assembly program. The previous example could be expanded in the following manner:

```
MACA  MAC   X, OPR, Y, Z
      LDA   (01)
      (02)  (04)
      STA   (03)
      ENDM
```

and used as

```
      FMB   P + 1
      MACA  P, SUB, P + 1, P + 2
      OTM   P + 1
```

The sequence would then be assembled as:

```
      FMB   P + 1, 1
      LDA   P
      SUB   P + 2
      STA   P + 1
      OTM   P + 1
```

The sequence of instructions defined as a macro is retained in its external BCD form (including all accompanying remarks) within the computer memory and is processed as if read from an external source each time the macro is used. It therefore behooves the programmer to omit unnecessary comments, lines and fields so as to conserve memory and allow for as many macros as possible.

The rules governing the use of macros in DAP are as follows:

1) All macro definitions must appear before the main body of the program.

2) Nested macros are not allowed.

3) The parameter substitution list is limited to eight fields.

4) The number and length of the macros used in a single program are restricted by the available memory size. (It can be seen that by judicious programming, one may make quite efficient use of the available memory size.)

5) Parameter substitutions into OCT, DEC, or BCI data fields are not allowed.

6) The pseudo-operation END may not appear in a macro skeleton.

MAC

The MAC (MACRO) pseudo-operation is used to define the start of a programmer-defined macro. The format for using the MAC pseudo-operation is:

LOCATION	The name of the MACRO
OPERATION	MAC
VARIABLE	One or more subfields, each containing a symbol (may be "dummy" parameters)
COMMENTS	Normal

The effect of the MAC pseudo-operation is to direct DAP to insert the subsequent instruction into the MACRO table.

ENDM

The ENDM (END OF MACRO) pseudo-operation is used to define the end of a programmer-defined MACRO. The format for using the ENDM pseudo-operation is:

LOCATION	Ignored
OPERATION	ENDM
VARIABLE	Ignored
COMMENTS	Normal

The effect of the ENDM pseudo-operation is to terminate the effect of the preceding MAC pseudo-operation.

STORAGE ALLOCATION PSEUDO-OPERATIONS

The pseudo-operations in this category (BES, BSS and COMN) are used for allocating storage for arrays of one or more elements.

BES

The BES (BLOCK ENDING WITH SYMBOL) pseudo-operation is used for reserving storage locations:

The format for using the BES pseudo-operation is:

LOCATION	Normal
OPERATION	BES
VARIABLE	Any absolute expression. Any symbol used in this field must have been previously defined.
COMMENTS	Normal

The effect of the BES pseudo-operation is to increase the value of the location counter by the value of the expression in the Variable Field. If there is a symbol in the Location Field, it is assigned the value of the Location Counter after the increase.

BSS

The BSS (BLOCK STARTING WITH SYMBOL) pseudo-operation is used for reserving storage locations. The format for using the BSS pseudo-operation is:

LOCATION	Normal
OPERATION	BSS
VARIABLE	Any absolute expression. Any symbol used in this field must have been previously defined.
COMMENTS	Normal

The effect of the BSS pseudo-operation is to increase the value of the Location Counter by the value of the expression in the Variable Field. If there is a symbol in the Location Field, it is assigned the value of the Location Counter before the increase.

COMN

The COMN (COMMON) pseudo-operation is used for absolutely assigning storage locations in upper memory. The format for using the COMN pseudo-operation is:

LOCATION	Normal
OPERATION	COMN
VARIABLE	Any absolute expression. Any symbol used in this field must have been previously defined.
COMMENTS	Normal

The effect of the COMN pseudo-operation is to cause DAP to subtract the value of the expression in the Variable Field from the COMMON base and assign this value to the symbol in the Location Field. COMMON base is a user option, but is assumed to be the address of the last memory location in the standard version of DAP. The COMN pseudo-operation establishes a common data "pool" that may be referenced by several programs.

SYMBOL DEFINING PSEUDO-OPERATION

The pseudo-operation in this category (EQU) is used for assigning an absolute or relocatable value to a symbol.

EQU

The EQU (EQUALS) pseudo-operation is used for defining a value for a symbol that is referred to by other DAP operations. The format for using the EQU pseudo-operation is:

LOCATION	Normal
OPERATION	EQU
VARIABLE	Any absolute or relocatable expression. Any symbol used in this field must have been previously defined.
COMMENTS	Normal

The EQU pseudo-operation causes DAP to evaluate the Variable Field expression for value and mode and assigns the value and mode to the Location Field symbol.

SECTION IV

SUBROUTINES

GENERAL

A sizeable body of subroutines is available to the DAP programmer. These include floating point, double precision and transcendental functions. Other subroutines may be added to suit the requirements of a particular installation.

A list of DDP-24 subroutines and information concerning their use is included in the DEP and Utility Manual.

Subroutines are called by using the CALL pseudo-operation in the regular programming sequence. DAP automatically generates the correct machine language instruction for the link between the main program and the subroutine.

When a subroutine is called within a program, DAP adds the reference to a table called the "transfer vector" and replaces the CALL with the machine language instruction, JST* A, where A is the address of the "transfer vector" table entry. Subsequent CALLs to the same subroutine will use the same table entry. When the END pseudo-operation is encountered, the "transfer vector" is tested to see if a subroutine was called. If no subroutine has been called, the assembly is terminated. If one or more subroutines have been called, the assembler requests that the subroutine library be mounted. The library is then scanned and the subroutines are added as part of the main program. If a called subroutine contains a CALL to a subroutine not used by the main program, this additional subroutine is also extracted from the library and punched as part of the program.

Figure 3 shows the memory layout of a program containing CALL's to two subroutines; each subroutine in turn calls one additional subroutine. Let the first two subroutines be names SUB1 and SUB2; SUB1 calls SUB3 and SUB2 calls SUB4. The subroutine library, in this example, contains the subroutines in the order of: SUB4, SUB3, SUB2, SUB1.

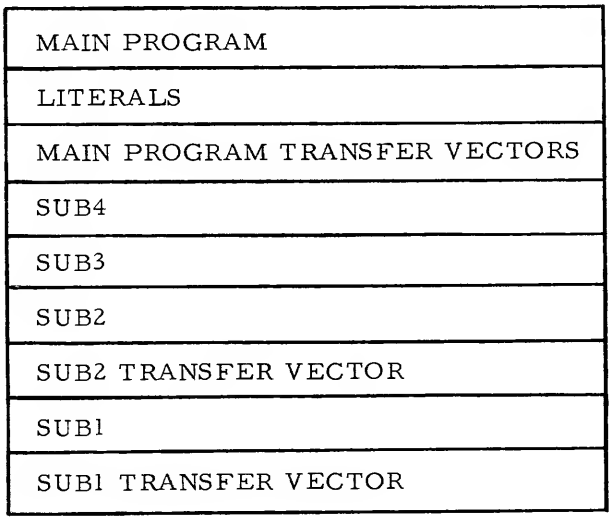


Figure 3. Memory Layout

SUBROUTINE LIBRARY

The subroutine library is contained on a punched paper tape, and is composed of two principal parts: 1) the library directory and 2) the subroutine programs.

LIBRARY DIRECTORY

The library directory consists of a set of tables defining each of the subroutines in the main body of the library. The tables contained in the directory are each two or more words in length. The first word contains the BCD name of the subroutine; the second word is divided into three fields -- these fields contain:

1) The number of computer words in the subroutine.

2) The number of erasable memory positions used by the subroutine.

3) A number, N, indicating any additional subroutines that are called by the current subroutine. N may be zero.

The remainder of the table is N words in length, where N is defined as in 3) above. Each word in this part of the table is a BCD subroutine name.

SUBROUTINE PROGRAMS

The program part of the subroutine library is made up of relocatable, assembled programs in a special format. This special format is supplied automatically by the assembly program when the pseudo-operation NTRY precedes a program to be assembled.

The first word of each program in the subroutine library is the BCD name of the subroutine. The second word is a parameter containing the number of erasable memory positions used by the subroutine in its address portion, and the number of COMN memory words used by the subroutine in the remainder of the word. With these specific exceptions, the relocatable subroutine programs are punched in the standard format.

UTILITY UPDATER

The Utility Updater may be used to duplicate, add to, or delete from the 3C subroutine library; control is exercised through the console typewriter. Stacked library modifications are possible; that is, one may make multiple insertions and/or deletions with a single pass over the library tape. The extent of this facility is limited only by the available memory size.

SUBROUTINE LIBRARY CHANGES

After the library updater has been called from the system tape, it will print on the typewriter:

ENTER INSTRUCTION:

The operator may then type in one of several instruction formats:

1) DUPLICATE

2) DELETE AAAA

3) INSERT AAAA AFTER BBBB

4) LIBRARY

DUPLICATE. After this instruction has been typed in, the machine will execute a HLT so that a paper tape may be mounted on the reader. Pressing the Program Start button on the console will then cause an identical copy of the input tape to be punched.

"DUPLICATE" will operate on an input tape of any format. The "DUPLICATE" operation will override any previous instructions that may have been given to the update routine.

DELETE AAAA. After this instruction has been typed in, the machine will store the information, return the typewriter carriage and again type:

ENTER INSTRUCTION:

INSERT AAAA AFTER BBBB, CCCC AFTER DDDD, ---. After this instruction, followed by a period, has been typed in, the computer will execute a HLT so that the tape containing the new subroutines may be mounted on the reader. Pressing the Program Start button on the console will then cause the mounted paper tape to be read. The desired routines will be extracted and stored, the typewriter carriage will be returned, and the machine will type:

Remaining Storage: XXXXX

ENTER INSTRUCTION:

Since one or more programs may be read from a single tape it is possible to merge libraries each of which contain several subroutines.

It is possible to insert a new subroutine as the first routine of the new library by entering the instruction as:

INSERT AAAA AFTER 0000,

using four zeros as the second name in the instruction.

Note that one need not merge all the subroutines from the input tape; it is possible to request only certain ones. Any others will be ignored.

LIBRARY. After this instruction has been typed in, the machine will execute a HLT so that the subroutine library may be mounted on the reader. Pressing the console Program Start button will cause the old library to be read and a new library with the requested modifications to be punched. The typewriter will list the subroutines in the order that they appear in the new library. The machine will then type:

LIBRARY UPDATE COMPLETED

and execute a HLT.

SECTION V

DAP OPERATING PROCEDURES

GENERAL

A DAP source program is prepared on the coding form previously described. It consists of a set of symbolic instructions and pseudo-operations terminated by an END pseudo-operation. (The pseudo-operation MOR is substituted under certain conditions; see below under sense switches.) The program is then punched into a paper tape called the source tape.

The utility tape is loaded into the tape reader and DAP is called into memory; the computer then halts so that the source tape may be mounted. Pressing the start button on the console then allows DAP to begin the assembly.

ASSEMBLER OPERATION

DAP is a two-pass assembly program; that is, the source program is fully scanned twice before the completed program is ready to load and execute. In general, this implies that the operator must manually remove the source program and replace it in the tape reader for the second pass. However, within the computer memory capabilities, it is possible for small programs to avoid this manual step by retaining the entire source program in memory. This is done automatically by DAP whenever possible.

When a source program calls for a subroutine, the operator must mount the subroutine library before the assembly can be completed (refer to Section IV).

SENSE SWITCHES

Most of the functions of DAP are automatic; however, certain options are available to the operator through the use of the computer console sense switches. The use of the sense switches is given in the following table.

S.S. No.	Condition	Meaning
1	Down	Listing is under program control.
	Up	All input is listed.
2	Down	Listing is under program control.
	Up	All listing is suppressed.
3	Down	Punching is normal.
	Up	Punching is suppressed.
4	Down	MOR pseudo-op is normal.
	Up	No halt on MOR.
5	Down	MOR pseudo-op is normal.
	Up	MOR pseudo-op = END.

ASSEMBLER PRODUCTS

Output from the DAP assembly program consists of a tabular on-line typewriter listing and a punched paper tape.

LISTING

The DAP printed output is called the assembly listing. It is a printing of the symbolic input instructions in the order in which they appeared together with the octal representation of the binary words produced by the assembler. A sample listing is shown in Figure 4.

The portion of the listing that is produced by the assembly appears on the left; the margin contains error symbols. The first column contains the line ID number -- an identification provided for the Source Program Update routine. The next column shows the location of each instruction; and, finally, in octal, the binary word assigned to the location. The machine operations are subdivided into separate subfields; numbers are given as eight octal digits with the appropriate sign; BCI words are eight digit logical groups.

The portion of the assembly listing appearing on the right is a copy of the original source program input. This part of the listing will be truncated on the right if the entire line exceeds the carriage capacity of the typewriter.

The assembler will indicate minor errors that occur in individual lines by inserting up to three flags in the left hand margin of the assembly listing (refer below to DIAGNOSTICS).

```
     001                                          ORG     320
     002    00500   +00000000        OCON        OCT     +0, 7777, -5
            00501   +00007777
            00502   -00000005
     003    00503   +00000012        XCON        DEC     +10, -4095
            00504   -00007777
     004    00505   0 1 56 77766     STRT        LDX     -10, 1
AX   005    00506   0 1 60 00503                 CRA     XCON, 1
     006    00507   0 1 10 01012     RUTN        ADD     TABL+12, 1
     007    00510   0 0 73 00515                 JOF     EXT2
     008    00511   0 1 75 00507                 JXI     RUTN, 1
     009    00512   0 1 54 00001                 ADX     1, 1
     010    00513   0 1 05 00507                 STA*    RUTN
     011    00514   0 0 74 00600     EXT1        JMP     RETN
     012    00515   0 0 74 00700     EXT2        SMP     EROR
     013                    00505                END     STRT
```

Note: It is assumed that the symbols TABL, RETN and EROR have been defined elsewhere in the program.

Figure 4. Sample Listing

PUNCHED TAPE

The paper tape output is a binary representation of the source program. It contains essentially the same information as is found on the left side of the assembly listing. Figure 5 shows the format of the paper tape records.

The first word of a paper tape record is an identification word containing the program name. This is followed by a variable number of information blocks. The first three words of each block are parameters. The first contains three fields:

1) An indicator of the type of information contained in the block.

2) A count of the number of program words in the block.

3) The computer address of the first program word.

The second and third words contain the relocation specifiers for the address fields of each word in the block (see below). The parameter words are followed by a sequence of up to 24 programwords, the number being determined by the count field in the first parameter word. The last word in each block is a check sum of the three parameter words and the program words.

Four relocation options are available to specify the processing of the address field of each computer word within a block. They are:

1) 00 the address is absolute

2) 01 the address is relocatable and positive

3) 10 the address is relocatable and negative

4) 11 not defined

DIAGNOSTICS

Certain diagnostic procedures are provided by the assembly program. These are divided into two categories, major and minor.

Major errors are those which will result in a final assembly that is either impossible or exceedingly difficult to correct at load time. These errors will be recorded on the on-line typewriter as they are encountered; the assembler will then execute a halt and the program may be continued or terminated at the discretion of the operator. Major errors are:

1) Symbol table capacity exceeded.

2) Macro table capacity exceeded.

3) Major pseudo-operation undefinable (i.e., an undefined symbol appearing in the variable field of a BES, BSS, COMN, EQU or ORG pseudo-operation).

Minor errors will be indicated by flags appended to the object program versus source program listing (refer to Figure 4). Minor errors and their associated flags are:

1) U Undefined symbol

2) M Multiple defined symbol

3) A Address field missing where required or present where not significant

4) X Index field missing where required or present where not significant

5) O Undefined operation code

6) C Conversion of a constant has exceeded word size or caused floating point overflow

7) E Any detected error not classified above.

Minor errors in a field will result in that field being assembled as zero except in the cases of multiple definitions where the last label definition is used, and constants where either a truncated or maximum value is supplied.

P R O G	Program Name (on subroutines only)
0 1 6 0 0 5 0 0	First parameter word
0 0 0 0 0 0 0 0	Relocation bits
0 0 0 0 0 0 0 0	
0 0 0 0 0 0 0 0	Body of program
0 0 0 0 7 7 7 7	
4 0 0 0 0 0 0 5	
0 0 0 0 0 0 1 2	
4 0 0 0 7 7 7 7	
1 5 6 7 7 7 6 6	
1 6 0 0 0 5 0 3	
1 1 0 0 1 0 1 2	
0 7 3 0 0 5 1 5	
1 7 5 0 0 5 0 7	
1 5 4 0 0 0 0 1	
4 0 5 0 0 5 0 7	
0 7 4 0 0 6 0 0	
0 7 4 0 0 7 0 0	
3 1 4 0 4 4 2 7	Check sum
1 0 0 0 0 5 0 5	END JUMP

Figure 5. Paper Tape Format For Object Program

APPENDIX A

DDP-24 CHARACTERISTICS

THE COMPUTER

Program operation on the DDP-24 involves the use of various machine registers. Although the intent of this manual is to describe the facilities of the DAP assembly program, a discussion of these registers is included, since they affect programming. Functionally, the computer consists of four units:

1) Arithmetic unit

2) Control unit

3) Input-output unit

4) Memory unit

ARITHMETIC UNIT

The arithmetic unit consists of three registers, A, B, and Z. From a programming viewpoint, the A- and B-registers serve as the arithmetic accumulator. The results of additions and subtractions are available in the A-register. Multiplication, resulting in a double length answer, uses both the A- and B-registers to contain the product, with the high order portion of the product in the A-register. Division, resulting in two answers (quotient and remainder), uses both the A- and B-registers, with the quotient in the B-register and the remainder in the A-register. The Z-register is unavailable to the programmer.

CONTROL UNIT

The control unit consists of the program counter, the index register, and other devices such as the shift counter and control clock. The program counter contains the memory address of the next instruction to be performed. Normally, its contents are incremented with 1 each time a command is executed; however, in the case of jump instructions, the contents of the program counter are replaced with the memory location of the jump destination. The index register is fully described below.

INPUT-OUTPUT UNIT

The DDP-24 provides a wide variety of input-output functions, all fully buffered for optimum machine utilization. A description of I-O programming options will be found in a later Appendix.

MEMORY UNIT

The memory unit of the DDP-24 uses ferrite cores as the storage device. Each core is referred to as a bit; core storage is divided into word units of 24 bits each. The basic memory consists of 4,096 such words.

Data items are stored in the computer words as absolute quantities with an associated arithmetic sign:

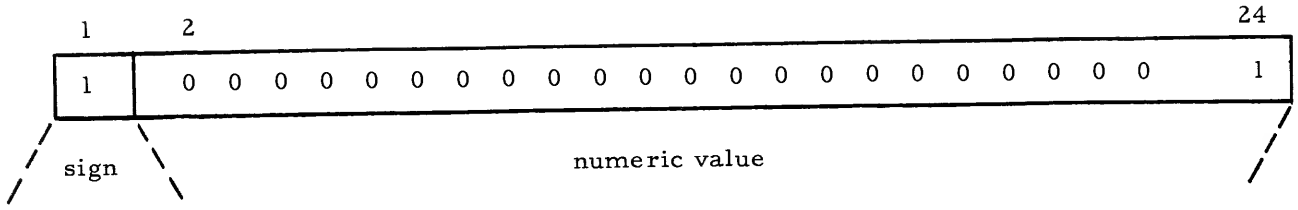


Figure 1

The word illustrated contains a minus one (-1).

The word, when used to contain a computer instruction, is subdivided, logically, into four areas:

1) instruction code
2) data address
3) index register assignment
4) indirect address specification

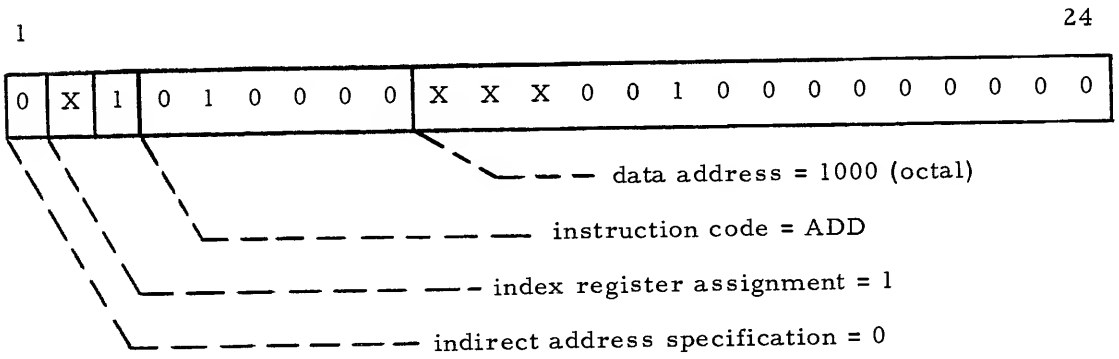


Figure 2

The word illustrated contains the instruction: ADD 1000,1.

The Xs indicate bit positions used by systems with expanded memory and indexing facilities.

The bit numbering convention established is from 1 through 24 from the left-most bit to the right-most. Therefore, Figure 2 can be summarized as follows:

Bit Number	Function
1	Indirect Address Bit
2-3	Index Register Specification
4-9	Operation Code
10-24	Address Portion of Word

In addition to the basic units and their associated registers, there are several sensing devices available to the programmer.

1) Overflow indicator.

The overflow indicator is turned on when an arithmetic operation gives a result that overflows the capacity of the A-register. Normally, this indicator, when set, remains on until tested by a machine instruction or is manually reset; however, multiple precision operations, when processed successfully, will leave the overflow indicator off regardless of its prior state.

2) Improper divide indicator.

The improper divide indicator is turned on when an attempt to divide results in a quotient exceeding the capacity of the B-register.

3) Sense switches.

There are 6 manually controlled switches at the operator's console. The status of these switches may be tested by the program.

4) The various I-O devices are provided with indicators that may be interrogated by the program.

INSTRUCTION CODES

The six bits of the instruction subfield and, in some cases, specified bits within the address subfield, make up the command structure of the DDP-24 computer. This command structure includes a full multiply and divide command, and a multiple precision step command for simple double, triple, and other multiple precision routines. For a description of each command and its uses, see Appendix B.

INDEX REGISTERS

The primary use of index registers is as instruction address modifiers. For example, the instruction shown in Figure 2 contains a data address of 1000 (octal); it also has a ONE in the index register assignment field. The true (or effective) address for the data to be operated upon by this instruction would not be 1000 (octal), but would be the arithmetic sum of 1000 plus the contents of the index register.

A fuller treatment of indexing is given in the section on addressing. In addition, the complete effect of indexing for each instruction is explained in the section on instruction codes.

INDIRECT ADDRESSING

Indirect addressing allows the programmer to make use of data addresses stored in other memory locations. In effect, the data address portion of an instruction containing an indirect specification is not the address as given in the instruction, but is instead the address as given in the location referred to by the instruction.

The usefulness of this feature can be illustrated in an example where it is necessary to compute the location of a data item and then store this computed address as part of an instruction. If the item must be referenced by several instructions, it is often more convenient to reference it indirectly at one location than to store the computed address several times.

In the following example, X is the address of a data item, Y is a memory location containing the number "X" in its address field; the instructions in memory locations A and B use the item, X, as an operand:

```
X       OCT     1 2 3 4 5 6 7 0
          .
          .
          .
Y       PZE     X
          .
          .
          .
A       MPY*    Y
          .
          .
          .
B       ADD*    Y
          .
          .
          .
```

(The asterisk (*) indicates that the instruction contains an indirect address specification.)

Indirect addressing is effective for all machine instructions and takes precedence over all other logical features regardless of the ultimate interpretation of the instruction. The computer hardware processing of indirectly addressed instructions is as follows:

The four principal parts of the instruction word are separated into unique registers as shown in Figure 3.

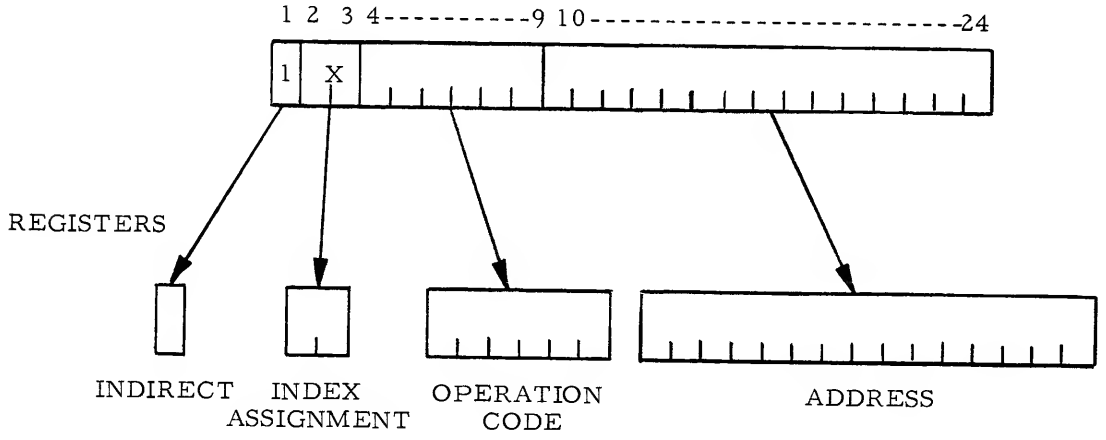


Figure 3

(Note: Do not confuse the index assignment register with the actual index register.)

EFFECTIVE ADDRESS

The effective address of the operand for a given instruction is dependent on three factors:

1) The contents of the address subfield

2) The contents of the index register subfield

3) The contents of the indirect address subfield

Case No. 1. Index register subfield and indirect address subfield both zero. The effective memory address is the number in the address subfield.

Case No. 2. Index register subfield non-zero and the indirect address subfield zero. The effective memory address is the arithmetic sum (truncated) of the contents of the memory address subfield and the contents of the specified index register.

Case No. 3. Index register subfield zero and the indirect address subfield non-zero. The effective memory address is found in the address portion of the memory location specified by the address subfield. (The contents of this remote memory location are interpreted using the same rules defined herein. That is, indirect addressing may be extended through several memory locations.)

Case No. 4. Index register subfield and indirect address subfields both non-zero. The effective address is derived by first computing the arithmetic sum of the contents of the address subfield and the specified index register, then using the contents of the address subfield of the computer memory location. (Further levels of indexing and indirect addressing as specified in Case No. 3 above are possible.)

APPENDIX B

DAP INSTRUCTION REPERTOIRE

The assembler translates all basic DDP-24 machine instructions. An instruction consists of the following:

1) A symbol or blanks in the Location Field.

2) The appropriate instruction mnemonic code in the Operation Field. The mnemonic code may be terminated by an asterisk (*) to denote indirect addressing.

3) Address and index assignment (separated by a comma) in the Variable Field. These may be written as expressions.

The following is a list of these instructions, their type, and the permissible fields. The codes used are:

R - Required

N - Not significant, but permissible

P - Permissible

Absence of a required item or presence of a non-significant item will result in an error flag on the appropriate line of the octal-symbolic listing.

Mnemonic	Octal Code	Address	Index
ADD	10	R	P
ADM	20	R	P
ADX	54	R	R
ALS	41	R	P
ANA	15	R	P
ARS	40	R	P
BCD	36	R	P
BIN	37	R	P
CRA	60	N	N
DIV	35	R	P
DMB	32	R	R
ENBI	51	N	N
ERA	17	R	P
FMB	31	R	R

Mnemonic	Octal Code	Address	Index
HLT	00	N	N
IAB	57	N	N
INA	52	P	N
INAM	52	P	N
INHI	51	N	N
INM	07	R	P
IRX	67	R	P
ITC	51	R	N
JIX	72	R	R
JMP	74	R	P
JOF	73	R	P
JPL	70	R	P
JRT	25	R	P
JST	27	R	P
JXI	75	R	R
JZE	71	R	P
LDA	24	R	P
LDB	23	R	P
LDX	56	R	R
LGL	47	R	P
LLR	43	R	P
LLS	45	R	P
LRR	42	R	P
LRS	44	R	P
MPY	34	R	P
NOP	77	P	P
NRM	46	N	R
OCP	53	R	N
ORA	16	R	P
OTA	50	P	N
OTAM	50	P	N
OTM	22	R	P

Mnemonic	Octal Code	Address	Index
RND	62	N	N
SBM	21	R	P
SCL	65	R	R
SCR	64	R	R
SKG	12	R	P
SKN	13	R	P
SKS	61	R	P
SMP	30	R	P
STA	05	R	P
STB	03	R	P
STC	04	R	P
STD	06	R	P
STX	66	R	R
SUB	11	R	P
TAB	55	N	N
TAX	63	N	R
XEC	02	R	P

SPECIAL MNEMONIC CODES

Three special mnemonic codes are provided for the convenience of the programmer when writing special instruction groups or calling sequences. They are assembled like any machine language instruction in that they may have address, index, and indirect fields.

Mnemonic	Address	Index	Assembles as
PZE	P	P	ZEROs in op-code
***	P	P	ZEROs in op-code
MZE	P	P	ZEROs in op-code and ONE in sign position

The following pseudo-operations are provided:

Operation Code	Purpose
ABS	Set punching format
BCI	Generate data
BES	Allocate storage
BSS	Allocate storage
CALL	Link subroutines
COMN	Allocate storage
DEC	Generate data
END	Assembly control
ENDM	Macro definition
EQU	Define symbols
LIST	Control printing
MAC	Macro definition
MOR	Assembly input control
NLST	Control printing
NTRY	Subroutine definition
OCT	Generate data
ORG	Allocate storage
REL	Set punching format

COMMAND INSTRUCTIONS

LOAD AND STORE INSTRUCTIONS

Code	Mnemonic	Execution Time	Description
03	STB	10 μsec	Store B The contents of B replace the contents of the memory word at the effective address. The contents of B are unchanged.
04	STC	10 μsec	Store Command Portion of A The contents of A, bits 1-9, replace the contents of the memory word, bits 1-9, at the effective address. The contents of A and the address portion of the memory word, bits 10-24, are unchanged.
05	STA	10 μsec	Store A The contents of A replace the contents of the memory word at the effective address. The contents of A are unchanged.

LOAD AND STORE INSTRUCTIONS (continued)

Code	Mnemonic	Execution Time	Description
06	STD	10 μsec	Store Address Portion of A The contents of A, bits 10-24, replace the contents of the memory word, bits 10-24, at the effective address. The contents of A and the op-code portion of the memory word, bits 1-9, are unchanged.
23	LDB	10 μsec	Load B The contents of the memory word at the effective address replace the contents of B. The contents of the memory word are unchanged.
24	LDA	10 μsec	Load A The contents of the memory word at the effective address replace the contents of A. The contents of the memory word are unchanged.
55	TAB	5 μsec	Transfer A to B The contents of A replace the contents of B. The contents of A are unchanged. The address portion and index bit of this instruction, bits 11-24 and 3, are not interpreted.
57	IAB	10 μsec	Interchange A and B The contents of A and B are interchanged. The address portion and index bit of this instruction, bits 11-24 and 3, are not interpreted.
60	CRA	5 μsec	Clear A The contents of A, bits 1-24, are set to zero. The address portion and index bit of this instruction, bits 11-24 and 3, are not interpreted.

ARITHMETIC INSTRUCTIONS

10	ADD	10 μsec	Add The contents of the memory word at the effective address are algebraically added to the contents of A, and the resultant sum replaces the contents of A. Overflow is possible and will set the overflow indicator. If the magnitude of the result is zero, the initial sign of A in unchanged. The contents of B and the memory word are unchanged.
11	SUB	10 μsec	Subtract The contents of the memory word at the effective address are algebraically subtracted from the contents of A, and the resultant difference replaces the contents of A. Overflow is possible and will set the overflow indicator. If the magnitude of the result is zero, the initial sign of A is unchanged. The contents of B and the memory word are unchanged.

Code	Mnemonic	Execution Time	Description
20	ADM	10 μsec	Add Magnitude The magnitude of the contents of the memory word at the effective address are added to the contents of A, and the resultant sum replaces the contents of A. The sign of the memory word is ignored; if the sign of A is negative, a subtractive process will occur. Overflow is possible and will set the overflow indicator. If the magnitude of the result is zero, the initial sign of A is unchanged.
21	SBM	10 μsec	Subtract Magnitude The magnitude of the contents of the memory word at the effective address are subtracted from the contents of A, and the resultant difference replaces the contents of A. The sign of the memory word is ignored; if the sign of A is negative, an add will occur. Overflow is possible and will set the overflow indicator. If the magnitude of the result is zero, the initial sign of A is unchanged. The contents of B and the memory word are unchanged.
30	SMP	10 μsec	Step Multiple Precision The contents of the memory word at the effective address are added to or subtracted from A such that the result has the sign of the result of the overall multiple precision operation. This sign and the selection of either add or subtract is determined by the instruction executed prior to the SMP instruction. Normally, this will take place at the beginning of a multiple precision routine. The add or subtract operation is for initial set-up of the multiple precision routine only; the sum or difference is not to be used further. For a multiple precision add, an ADD operation of the highest order portion of the two operands will be followed by SMP instructions which add all portions, starting with the lowest order and producing the same signs. For a multiple precision subtract, a SUB operation of the highest order portion of the two operands will be followed by SMP instructions which subtract all portions, starting with the lowest order and producing the same signs. Any carry (or borrow) produced by an SMP step will be properly added (or subtracted) at the following SMP. Overflow is set by the SMP command if a carry is produced; overflow is reset if no carry is produced. Overflow of a multiple precision addition or subtraction can be detected by checking the overflow indicator after completion of the operation (normally it would not be set after the last SMP operation).
34	MPY	31 μsec	Multiply The contents of B are multiplied by the contents of the memory word at the effective address. The 23 most significant bits of the 46-bit product replace the contents of A, bits 2-24; the least significant bits replace

ARITHMETIC INSTRUCTIONS (continued)

Code	Mnemonic	Execution Time	Description
	MPY (cont)		the contents of B, bits 2-24. The signs of A and B are set to the algebraic sign of the product. The contents of A are cleared at the start of this instruction. The contents of the memory word are unchanged. The B-register must be loaded prior to the execution of the MPY instruction.
35	DIV	33 μsec	Divide The contents of the memory word at the effective address (the divisor) are divided into the contents of both A and B (the double-length dividend). The 23-bit quotient replaces the contents of B, bits 2-24; the 23-bit remainder (absolute value) replaces the contents of A, bits 2-24. The signs of A and B are set to the algebraic sign of the quotient. If the initial magnitude of A is equal to or greater than the magnitude of the memory word, the improper divide indicator is set. The contents of the memory word are unchanged.
36	BCD*	33 μsec	BCD to Binary Conversion The contents of the memory word at the effective address are converted from BCD into binary, the result replaces the contents of A. The contents of B are destroyed; the contents of the memory word are unchanged. The maximum BCD number which can be converted with this instruction is decimal $\pm$799,999.
37	BIN*	33 μsec	Binary to BCD Conversion The contents of the memory word at the effective address are converted from binary to BCD code; the result replaces the contents of B. The conversion will be performed only on those bits of the memory word which will produce a BCD code within the capacity of the B-register (24 bits). The improper divide indicator will be set if the binary number to be converted is greater than octal 3,032,377, resulting in a BCD number greater than decimal 799,999. The contents of A are destroyed; the contents of the memory word are unchanged.
62	RND	6 μsec	Round A The contents of A are incremented by one if bit 2 in the B-register is a one; the contents of A are unchanged if bit 2 (in B) is a zero. The address portion and index bit of this instruction, bits 11-24 and 3, are not interpreted. Overflow is possible and will set the overflow indicator. The contents of B remain unchanged.

* Denotes optional command

LOGICAL INSTRUCTIONS

Code	Mnemonic	Execution Time	Description
15	ANA	10 μsec	AND to A This instruction forms the logical product of the contents of A and the contents of the memory word at the effective address and replaces the contents of A with the result. For each ZERO in the contents of the memory word, a ZERO is written into the corresponding bit in A; for each ONE in the memory word, the corresponding bit in A is unchanged. The contents of B and the memory word are unchanged.
16	ORA	10 μsec	OR to A This instruction forms the logical sum of the contents of A and the contents of the memory word at the effective address and replaces the contents of A with the result. For each ONE in the contents of the memory word, a ONE is written into the corresponding bit in A; for each ZERO in the memory word, the corresponding bit in A is unchanged. The contents of B and the memory word are unchanged.
17	ERA	10 μsec	Exclusive OR to A This instruction forms the logical exclusive sum of the contents of A and the contents of the memory word at the effective address and replaces the contents of A with the result. For each ONE in the contents of the memory word, the corresponding bit in A is complemented; for each ZERO in the memory word, the corresponding bit in A is unchanged. The contents of B and the memory word are unchanged.
SHIFT INSTRUCTIONS			
40	ARS	5+n μsec	A Right Shift The contents of A, bits 2-24, are shifted to the right the number of positions specified by the six least significant bits of the instruction, bits 19-24. The sign of A is not shifted and is unchanged. ZEROs are shifted into the vacated position next to the sign of A, bit 2; bits shifted out of the low order position are lost. This instruction may be indexed, in which case the number of shift steps is the sum of the address portion of the instruction and the contents of the index register. The contents of B are unchanged.
41	ALS	5+n μsec	A Left Shift The contents of A, bits 2-24, are shifted to the left the number of positions specified by the six least significant bits of the instruction, bits 19-24. The sign of A is not shifted and is unchanged. ZEROs are shifted into the vacated low order position of A; bits shifted out of the position next to the sign of A (bit 2) are lost. This instruction may be indexed, in which case the number of shift steps is the sum of the address portion of the instruction and the contents of the index register. The contents of B are unchanged.

SHIFT INSTRUCTIONS (continued)

Code	Mnemonic	Execution Time	Description
42	LRR	5+n μsec	Long Right Rotate The contents of A, bits 1-24 and B, bits 1-24, are treated as a single 48-bit register and are rotated to the right (end around carry) the number of positions specified by the six least significant bits of the instruction, bits 19-24. The signs of A and B are also shifted. Bits shifted out of the low order position of A enter the high order position of B; bits shifted out of the low order position of B enter the high order position of A. This instruction may be indexed, in which case the number of shift steps is the sum of the address portion of the instruction and the contents of the index register.
43	LLR	5+n μsec	Long Left Rotate The contents of A, bits 1-24, and B, bits 1-24, are treated as a single 48-bit register and are rotated to the left (end around carry) the number of positions specified by the six least significant bits of the instruction, bits 19-24. The signs of A and B are also shifted. Bits shifted out of the high order position of B enter the low order position of A; bits shifted out of the high order position of A enter the low order position of B. The instruction may be indexed, in which case the number of shift steps is the sum of the address portion of the instruction and the contents of the index register.
44	LRS	5+n μsec	Long Right Shift The contents of A, bits 2-24, and B, bits 2-24, are treated as a single 46-bit register and are shifted to the right the number of positions specified by the six least significant bits of the instruction, bits 19-24. The signs of A and B are not shifted; however, the sign of B is made to agree with the sign of A. ZEROs are shifted into the vacated position next to the sign of A, bit 2; bits shifted out of the low order position of A enter the position next to the sign of B, bit 2. Bits shifted out of the low order position of B are lost. This instruction may be indexed, in which case the number of shift steps is the sum of the address portion of the instruction and the contents of the index register.
45	LLS	5+n μsec	Long Left Shift The contents of A, bits 2-24, and B, bits 2-24, are treated as a single 46-bit register and are shifted to the left the number of positions specified by the six least significant bits of the instruction, bits 19-24. The signs of A and B are not shifted; however, the sign of A is made to agree with the sign of B. ZEROs are shifted into the vacated low order position of B; bits shifted out of the position next to the sign of B, bit 2, enter the low order position of A. Bits shifted out of the position next to the sign of A, bit 2, are lost. This instruction may be indexed, in which case the number of shift steps is the sum of the address portion of the instruction and the contents of the index register.

SHIFT INSTRUCTIONS (continued)

Code	Mnemonic	Execution Time	Description
46	NRM	5+n μsec	Normalize The contents of A, bits 2-24, and B, bits 2-24, are treated as a single 46-bit register and are shifted left until a ONE is shifted into the position next to the sign of A, bit 2, or until the contents of B replace the contents of A (46 steps). If the index position, bit 3, is a ONE, the number of shifts required for normalization is subtracted from the index register. If the index position, bit 3, is a ZERO, the index register is not affected by this instruction. ZEROs are shifted into the vacated low order position of B; bits shifted out of the position next to the sign of B, bit 2, enter the low order position of A. If a ONE is in the position next to the sign of A, bit 2, at the start of the operation (already normalized), the instruction will be treated as a NOP. The signs of A and B are not shifted and are unchanged.
47	LGL	5+n μsec	Logical Left Shift The contents of A, bits 1-24, are shifted to the left the number of positions specified by the six least significant bits of the instruction, bits 19-24. The sign of A is also shifted. ZEROs are shifted into the vacated low order position of A; bits shifted out of the high order position of A are lost. This instruction may be indexed, in which case the number of shift steps is the sum of the address portion of the instruction and the contents of the index register. The contents of B are unchanged.
64	SCR	5+n μsec	Scale Right The contents of A, bits 2-24, and B, bits 2-24, are treated as a single 46-bit register and are shifted to the right the number of positions specified by the six least significant bits of the instruction, bits 19-24. The number of positions shifted will be added to the contents of the index register. The signs of A and B are not shifted; however, the sign of B is made to agree with the sign of A. ZEROs are shifted into the vacated position next to the sign of A, bit 2; bits shifted out of the low order position of A enter the position next to the sign of B, bit 2. Bits shifted out of the low order position of B are lost. This instruction is not valid if the index position, bit 3, is a ZERO.
65	SCL	5+n μsec	Scale Left The contents of A, bits 2-24 and B, bits 2-24, are treated as a single 46-bit register and are shifted to the left the number of positions specified by the six least significant bits of the instruction, bits 19-24. The number of positions shifted will be subtracted from the contents of the index register. The signs of A and B are not shifted; however, the sign of A is made to agree with the sign of B. ZEROs are shifted

SHIFT INSTRUCTIONS (continued)

Code	Mnemonic	Execution Time	Description
	SCL (cont)		into the vacated low order position of B; bits shifted out of the position next to the sign of B, bit 2, enter the low order position of A. Bits shifted out of the position next to the sign of A, bit 2, are lost. This instruction is not valid if the index position, bit 3, is a ZERO.
JUMP INSTRUCTIONS			
12	SKG	10 or 12 μsec	Skip if A Greater The contents of A are algebraically compared to the contents of the memory word at the effective address. If the value in A is greater than the value in the memory word, the next instruction is skipped and the computer resumes at that point. If the value in A is equal to or less than the value in the memory word, the computer takes the next sequential instruction. The contents of A and the memory word are unchanged.
13	SKN	10 or 12 μsec	Skip if A Not Equal The contents of A are algebraically compared to the contents of the memory word at the effective address. If the value in A is not equal to the value in the memory word, the next instruction is skipped and the computer resumes at that point. If the value in A is equal to the value in the memory word, the computer takes the next sequential instruction. The contents of A and the memory word are unchanged.
25	JRT	10 μsec	Jump Return This instruction is an indirect jump. A jump is executed to the location specified by the address portion of the memory word (bits 11-24), at the effective address. The contents of the memory word and the contents of A and B are unchanged. This instruction must be used for returning from all interrupt subroutines to restore the interrupt capability again (in the standard DDP-24), or to effect return to the previous priority level if optional Priority Interrupt is used.
27	JST	10 μsec	Jump and Store Location The location of the JST instruction plus one replaces the contents of the address portion of the memory word, bits 11-24, at the effective address. A jump is then executed to one location beyond the effective address. Bits 1-10 of the contents of the memory word are unchanged; the contents of A and B are unchanged. This instruction may be used for entering a subroutine.
70	JPL	6 μsec	Jump if A Plus If the sign of A, bit 1, is positive (ZERO), the computer takes its next instruction from the memory word at the effective address and continues from there. If the sign of A, bit 1, is negative (ONE),

JUMP INSTRUCTIONS (continued)

Code	Mnemonic	Execution Time	Description
	JPZ (cont)		the computer takes the next sequential instruction. Thus, a jump for A negative could be accomplished by an unconditional jump instruction (JMP) immediately following the JPL instruction. The contents of A are unchanged.
71	JZE	5 μsec	Jump if A Zero If all of the magnitude positions in A, bits 2-24 are ZEROs, the computer takes its next instruction from the memory word at the effective address and continues from there. If any of the magnitude positions of A are ONEs, the computer takes the next sequential instruction. The sign of A, bit 1, is ignored. The contents of A are unchanged.
73	JOF	5 μsec	Jump on Overflow If the overflow indicator is set, it will be reset and the computer will take its next instruction from the memory word at the effective address and continue from there. If the overflow indicator is not set, the computer takes the next sequential instruction. To reset the overflow indicator without altering the normal sequence of instructions, the JOF instruction may be used with an effective address that is one location greater than the address of the JOF instruction.
74	JMP	5 μsec	Unconditional Jump The computer takes its next instruction from the memory word at the effective address and continues from there. The JMP instruction with indirect addressing may be used for returning from subroutines which are not interrupt routines.

INDEX INSTRUCTIONS

54	ADX	5 μsec	Add to Index The contents of the address portion of this instruction, bits 11-24, are added to the contents of the index register, and the resultant sum replaces the contents of the index register. Overflow of the index register is possible, but will be ignored. If the indirect address position of the instruction, bit 1, is a ONE, the contents of the address portion of the memory word, bits 11-24, at the effective address are added to the index register. The contents of A and the memory word are unchanged. This instruction is not valid if the index position of the instruction, bit 3, is a ZERO.

INDEX INSTRUCTIONS (continued)

Code	Mnemonic	Execution Time	Description
56	LDX	5 μsec	Load Index The contents of the address portion of this instruction, bits 11-24, replace the contents of the index register. If the indirect address position of the instruction, bit 1, is a ONE, the contents of the address portion of the memory word, bits 11-24, at the effective address replace the contents of the index register. The contents of A or the memory word are unchanged. This instruc- is not valid if the index position of the instruction, bit 3, is a ZERO.
63	TAX	5 μsec	Transfer A to Index The address portion of A, bits 11-24, replace the contents of the index register. The contents of A are unchanged. This instruction is not valid if the index position of the instruction, bit 3, is a ZERO. The indirect address position and the address portion of this instruction, bits 1 and 11-24, are not interpreted.
66	STX	10 μsec	Store Index The contents of the index register replace the contents of the address portion of the memory word, bits 11-24, at the effective address. The contents of A and the index are unchanged; bits 1-10 of the memory word are unchanged. This instruction is not valid if the index position of the instruction, bit 3, is a ZERO.
67	IRX	14 μsec	Increment, Replace and Load Index The contents of the address portion of the memory word, bits 11-24, at the effective address are incremented by one. If the index position of this instruction, bit 3, is a ONE, the resulting sum replaces the contents of the address portion of the memory word and the index register. The contents of A are unchanged. If the index position of this instruction, bit 3, is a ZERO, the address portion of the memory word will be incremented and this incremented value will replace the contents of A. In this case, the index register contents are unchanged. Any carry from bit 11 is ignored. Bits 1-10 of the memory word are unchanged. Thus, it is possible to have many "index registers" in memory that can be incremented, saved and made available for use in indexing operations all with one instruction.
72	JIX	5 μsec	Jump on Index If the contents of the index register are not zero, the computer takes its next instruction from the memory word at the effective address and continues from there. If the contents of the index register are zero, the computer takes the next sequential instruction. This instruction is not valid if the index position of the instruction, bit 3, is a ZERO.

INDEX INSTRUCTIONS (continued)

Code	Mnemonic	Execution Time	Description
75	JXI	7 μsec	Jump on Index Incremented The contents of the index register are incremented by one and the resulting sum replaces the contents of the index register. If this resulting sum is not zero, the computer takes its next instruction from the memory word at the effective address and continues from there. If the sum is zero, the computer takes the next sequential instruction. This instruction is not valid if the index position of the instruction, bit 3, is a ZERO.

INPUT-OUTPUT INSTRUCTIONS

07	INM	10 μsec	Input to Memory The input word from a previously enabled input channel (refer to OCP) replaces the contents of the memory word at the effective address.
22	OTM	10 μsec	Output from Memory The contents of the memory word at the effective address are transferred as output to the previously enabled output channel (refer to OCP). The contents of the memory word are unchanged.
31	FMB	variable	Fill Memory Block This instruction is used for high speed input into the block of consecutive memory locations starting with the memory word at the effective address. Once started, the sequence continues without interruption, controlled asynchronously by an external ready signal. The FMB instruction may operate with any input channel that has been previously enabled (refer to OCP). The contents of the index register are incremented by one for each word being stored thereby increasing the effective address. Execution of this instruction may be terminated by either an external signal (e.g., a stop code) or upon the contents of the index register having become all ZEROs. This instruction is not valid if the index position, bit 3, is a ZERO. The FMB instruction can process input data at a 166 kc word rate.
32	DMB	variable	Dump Memory Block This instruction is used for high speed output from the block of consecutive memory locations starting with the memory word at the effective address. Once started, the sequence continues without interruption, controlled asynchronously by an external ready signal. The DMB instruction may operate with any output channel that has been previously enabled (refer to OCP). The contents of the index register are incremented by one for each output word being transferred thereby increasing the effective address. Execution of this instruction may be terminated by either an external signal or upon the contents of the index

Code	Mnemonic	Execution Time	Description
	DMB (cont)		register having become all ZEROs. This instruction is not valid if the index position, bit 3, is a ZERO. The DMB instruction can process output data at a 166 kc word rate.
50	OTA	5 μsec	Output from A The contents of the A register are transferred as output to the previously enabled output channel. If bit 11 of this instruction contains a ONE, bits 19-24 form a 6 bit mask. This provides a facility for flexible formatting of character outputs. For ZERO mask bits, there will be corresponding ZERO bits in the output; for ONE mask bits, the corresponding bits in A will be the output. If bit 11 contains a ZERO, 24-bit output words are transferred by this instruction. The index position of the instruction, bit 3, is not interpreted. The contents of A are unchanged.
51	ITC	5 μsec	Interrupt Control Interrupt is enabled if bit 11 of this instruction contains a ONE; interrupt is inhibited if bit 11 contains a ZERO. The index position and the remaining positions of the address portion of the instruction, bits 3 and 12-24, are not interpreted.
52	INA	5 μsec	Input to A The input word from a previously enabled input channel (refer to OCP) replaces the contents of A. If bit 11 of this instruction contains a ONE, bits 19-24 form a 6 bit mask. This provides a facility for flexible formatting of character inputs. For ZERO mask bits, there will be corresponding ZERO bits in A; for ONE mask bits, the corresponding input bits will replace the contents of A. If bit 11 contains a ZERO, 24-bit input words are transferred by this command. The index position of the instruction, bit 3, is not interpreted.
53	OCP	5 μsec	Output Control Pulse An output pulse is generated by this instruction for the control of input-output channels and external equipment. The address portion, bits 11-24, specifies the unit to be selected, the type of control, etc. (refer to APPENDIX C for the code assignments). The index position of the instruction, bit 3, is not interpreted.
61	SKS	5 μsec	Skip if Sense Line Not Set The sense line specified by the address portion of this instruction, bits 11-24, is interrogated. If the sense line is not set, the computer skips the next instruction and continues from there; if the sense line is set, the computer will take the next sequential instruction. The lines that may be tested include 10 internal sense lines (six sense switches, overflow indicator, improper divide indicator, input parity and

INPUT-OUTPUT INSTRUCTIONS (continued)

Code	Mnemonic	Execution Time	Description
	SKS (cont)		stop code), ready signals of input-output channels and external sense lines from peripheral equipment (busy status, parity errors, etc.) From one to ten of the internal sense lines may be tested simultaneously; in which case any or all of the tested sense lines may cause a skip. For the channel-ready sense lines, similar simultaneous testing is also possible. If the index position of the instruction, bit 3, is a ONE, the flip-flop associated with the tested sense line is reset. APPENDIX D contains the sense line selection assigments.
CONTROL INSTRUCTIONS			
00	HLT	5 μsec	Halt The computer will halt until the START button is manually depressed (see description of operation), at which time execution will be resumed at the next sequential instruction. The indirect address position, index position and address portion of this instruction, bits 1, 3 and 11-24, are not interpreted.
02	XEC	5 μsec + variable	Execute The instruction in the memory word at the effective address is executed. After execution of the specified instruction, the computer takes the next sequential instruction following the XEC instruction and continues from there. If the executed instruction involves a jump, the computer takes its next instruction from the jump destination and continues from there; if the executed instruction involves a skip, the skip will be relative to the XEC instruction and not the instruction at the effective address.
77	NOP	5 μsec	No Operation No operation is performed by this instruction. The computer will take the next sequential instruction and continue from there. The index position and address portion of this instruction, bits 3 and 11-24, are not interpreted.

APPENDIX C

OCP CONTROL PULSE CODES

The following is a list of assigned codes to be used in the address portion of the OCP instruction to perform the specified functions. The codes are given in octal notation.

OCP ADDRESS CODES (x = unit number)

BOTH INPUT AND OUTPUT CHANNELS

00000	Enable Standard I/O Character Channels
00001	Enable No. 1 Optional I/O Character Channels
thru	thru
00007	Enable No. 7 Optional I/O Character Channels
00010	Enable Standard I/O Word Channels
00011	Enable No. 1 Optional I/O Word Channels
thru	thru
00076	Enable No. 54 Optional I/O Word Channels
00077	Inhibit all I/O Channels

INPUT CHANNELS ONLY

00100	Enable Standard Input Character Channel
00101	Enable No. 1 Optional Input Character Channel
thru	thru
00107	Enable No. 7 Optional Input Character Channel
00110	Enable Standard Input Word Channel
00111	Enable No. 1 Optional Input Word Channel
thru	thru
00176	Enable No. 54 Optional Input Word Channel

OUTPUT CHANNELS ONLY

00200	Enable Standard Output Character Channel

OUTPUT CHANNELS ONLY (continued)

00201	Enable No. 1 Optional Output Character Channel
thru	thru
00207	Enable No. 7 Optional Output Character Channel
00210	Enable Standard Output Word Channel
00211	Enable No. 1 Optional Output Word Channel
thru	thru
00276	Enable No. 54 Optional Output Word Channel
00277	Inhibit all Output Channels

WORD BUFFER

003mx	Word Buffer Control (8 possible)
	m = number of characters per word

MISCELLANEOUS

01000	Enable Stop Code Punch
01001	General Purpose Control Pulses for External Devices
thru	thru
01777	01001 thru 01007 Standard with DDP-24

TYPEWRITER

0200x	Typewriter Input Select (Keyboard Enabled)
0201x	Typewriter Output Select (Keyboard Inhibited)
0207x	Typewriter Disconnect (Keyboard Released)
	Note: x = 0 corresponds to standard typewriter

PAPER TAPE READER

0210x	Paper Tape Reader Start
0217x	Paper Tape Reader Stop
	Note: x = 0 corresponds to standard paper tape reader

PAPER TAPE PUNCH

0220x	Paper Tape Punch Select
	Paper Tape Punch Disconnect
	Note: x = 0 corresponds to standard paper tape punch

LINE PRINTER

0230x

thru Card Reader Control

0237x

CARD READER

0240x

thru Card Reader Control

0257x

DIGITAL X Y PLOTTER CONTROL (2 possible)

026nn Plotter No. 1

027nn Plotter No. 2

nn =

01	Step -Y (carriage right)
02	Step +Y (carriage left)
04	Step -X (drum up)
05	Step -X, +Y
06	Step -X, +Y
10	Step +X (drum down)
11	Step +X, -Y
12	Step +X, +Y
20	Plotter pen down
40	Plotter pen up

MAGNETIC TAPE CONTROL (8 possible)

03x00 thru 03X17	Subselect one of up to 16 tape units connected to the same channel
03x23	Start tape handler and read one block, even parity (to be preceded by OCP instruction with either 03x42, 03x43, 03x52 or 03x53)
03x24	Start tape handler and write one block, even parity (to be preceded by OCP instruction with either 03x42 or 03x52)
03x25	Start tape handler and read one block, odd parity (to be preceded by OCP instruction with either 03x42, 03x43, 03x52 or 03x53 in the address portion)
03x26	Start tape handler and write one block, odd parity (to be preceded by OCP instruction with either 03x42 or 03x52 in address portion)

MAGNETIC TAPE CONTROL (continued)

03x41	Move one block (to be preceded by OCP instruction with either 03x42, 03x43, 03x52 or 03x53)
03x42	Forward with interrupt by record gaps
03x43	Reverse with interrupt by record gaps
03x44	Search for file gap
03x50	Stop transport
03x51	Start transport (to be preceded by either 03x42, 03x43, 03x52 or 03x53)
03x52	Forward
03x53	Reverse
03x54	Fast Forward
03x55	Fast Reverse
03x56	Rewind to load point
03x61	Reset write flip-flop

A/D AND D/A CONTROL

04000	
thru	A to D Control (to be specified)
04377	
04400	
thru	D to A Control (to be specified)
04777	

DIGITAL RESOLVER

05000	T mode with prescaling
05001	T mode without prescaling
05002	T* mode with prescaling
05003	T* mode without prescaling
05004	H mode with prescaling
05005	H mode without prescaling
05006	H* mode with prescaling
05007	H* mode without prescaling
05010	Sequential load of DR; start with Y
05011	Sequential load of DR; start with W
05012	Sequential load of DR; start with W

DIGITAL RESOLVER (continued)

05013	Sequential DR output; start with Y
05014	Sequential DR output; start with X
05015	Sequential DR output; start with W

CONTROL OF DIRECT MEMORY ACCESS AND FULLY BUFFERED CHANNELS

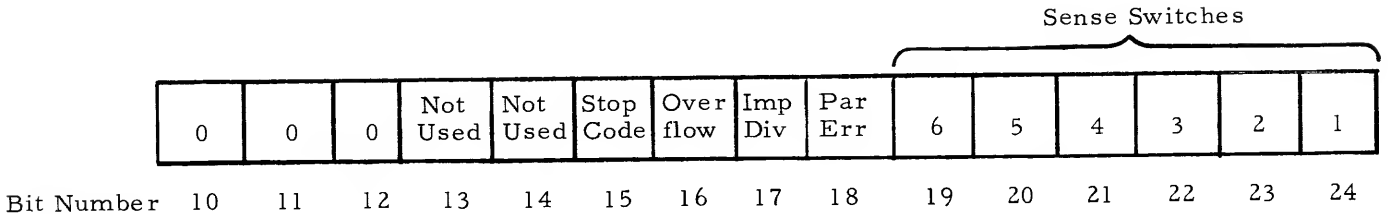
0600n	
thru	Direct Memory Access Channels
0603n	
0604n	
thru	Fully Buffered Channels
0607n	
	n = function specification

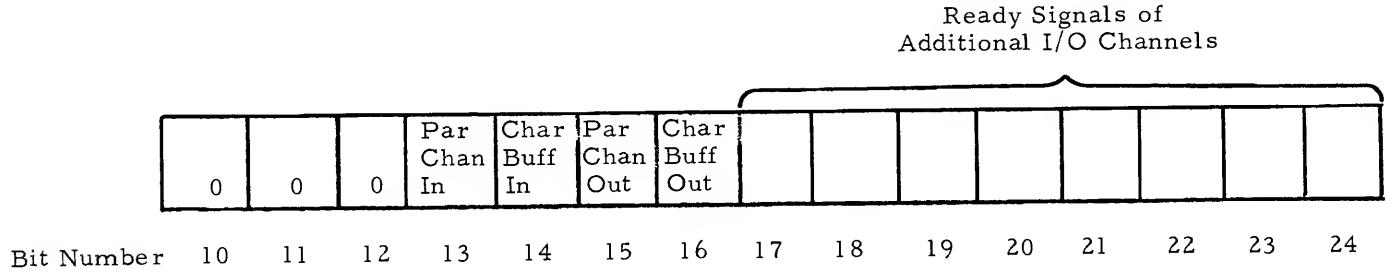
APPENDIX D

SKS SENSE LINE CODES

INTERNAL TESTS

The address portion of the SKS command has the following formats and meanings:





Simultaneous tests are possible. The results of all tests are OR'd together. For example, if 19 and 16 are one's, both sense switch 6 and the overflow flip-flop are tested. If either is set, no skip is generated.

EXTERNAL TESTS

The address portion of the SKS command specifies 16 sense lines as follows (octal code):

2XXX0 through 2XXX7 and

3XXX0 through 3XXX7

Bits 13-21 of the address are not normally used in the testing, but each of the 16 sense lines may actually represent a group of 512 different test signals when these bits are specified.

Note: If the index bit of the SKS command is set, any flip-flop tested by the SKS command is also reset by it.

Test	Octal Code
Sense Switch 6	00040
Sense Switch 5	00020
Sense Switch 4	00010
Sense Switch 3	00004
Sense Switch 2	00002
Sense Switch 1	00001
Parity Error	00100
Improper Divide	00200
Overflow	00400
Stop Code	01000

APPENDIX E

TYPEWRITER CODES

Lower Case	Upper Case	Octal
ø	b	00
1	■	01
2	■	02
3	■	03
4	:	04
5	@	05
6	✓	06
7	>	07
8	■	10
9	■	11
#	¬	13
¤	¢	20
/	■	21
S	■	22
T	■	23
U	=	24
V	%	25
W	"	26
X	'	27
Y	■	30
Z	■	31
,	■	33
-	-	40
J	■	41
K	■	42
L	■	43
M	)	44
N	¤	45
O	Δ	46
P	;	47
Q	■	50
R	■	51
tab		52
$	■	53

	Lower Case	Upper Case	Octal
1)	Backspace (stop)	Backspace (stop)	54
	Space	Space	56
2)	&	&	60
	A	■	61
	B	■	62
	C	■	63
	D	(	64
	E	¤	65
	F	‡	66
	G	<	67
	H	¬	71
	I	■	71
	•	~	73
	Shift LC	Shift LC	74
	Shift UC	Shift UC	75
	Carriage Return	Carriage Return	76
3)	Line Feed	Line	77

1) Backspace includes an eighth level punch on paper tape. The eighth level punch is an automatic stop code for the tape reader.

2) Lower case & is interpreted as + by DAP.

3) Octal 77 code is ignored by DAP in either case.

APPENDIX F

NUMERICAL INSTRUCTION LIST

Code	Mnemonic	Name	Page
00	HLT	Halt	B-16
02	XEC	Execute	B-16
03	STB	Store (B) in (EA)	B-4
04	STC	Store Command Portion of (A) in (EA)	B-4
05	STA	Store (A) in (EA)	B-4
06	STD	Store Address Portion of (A) in (EA)	B-5
07	INM	Input to Memory	B-14
10	ADD	Add (EA) to (A)	B-5
11	SUB	Subtract (EA) from (A)	B-5
12	SKG	Skip for (A) > (EA)	B-11
13	SKN	Skip for (A) ≠ (EA)	B-11
15	ANA	Logical AND to A	B-8
16	ORA	Logical OR to A	B-8
17	ERA	Logical Exclusive OR to A	B-8
20	ADM	Add Magnitude of (EA) to (A)	B-6
21	SBM	Subtract Magnitude of (EA) from (A)	B-6
22	OTM	Output from Memory	B-14
23	LDB	Load B with (EA)	B-5
24	LDA	Load A with (EA)	B-5
25	JRT	Jump Return	B-11
27	JST	Jump Store	B-11
30	SMP	Step Multiple Precision	B-6
31	FMB	Fill Memory Block	B-14
32	DMB	Dump Memory Block	B-14
34	MPY	Multiply	B-6
35	DIV	Divide	B-7
*36	BCD	BCD to Binary Conversion	B-7
*37	BIN	Binary to BCD Conversion	B-7
40	ARS	A Register Right Shift	B-8
41	ALS	A Register Left Shift	B-8
42	LRR	Long Right Rotate A and B Registers	B-9
43	LLR	Long Left Rotate A and B Registers	B-9
44	LRS	Long Right Shift A and B Registers	B-9

* Denotes optional instruction

Code	Mnemonic	Name	Page
45	LLS	Long Left Shift A and B Registers	B-9
46	NRM	Normalize Shift A and B Registers	B-10
47	LGL	Logical Left Shift A Register	B-10
50	OTA	Output from A	B-15
50	OTAM	Output from A Masked	B-15
51	ITC	Interrupt Control	B-15
51	ENBI	Enable Interrupt	B-15
51	INHI	Inhibit Interrupt	B-15
52	INA	Input to A	B-15
52	INAM	Input to A Masked	B-15
53	OCP	Output Control Pulse	B-15
54	ADX	Add to Index Register	B-12
55	TAB	Transfer (A) to B	B-5
56	LDX	Load Index Register	B-13
57	IAB	Interchange (A) and (B)	B-5
60	CRA	Clear A	B-5
61	SKS	Skip for Sense Line Not Set	B-15
62	RND	Round (A)	B-7
63	TAX	Transfer Address Portion of A Register to Index Register	B-13
64	SCR	Scale Right A and B Registers	B-10
65	SCL	Scale Left A and B Registers	B-10
66	STX	Store Index Register	B-10
67	IRX	Increment, Replace, and Load Index Register	B-13
70	JPL	Jump on (A) Plus	B-11
71	JZE	Jump on (A) Zero	B-12
72	JIX	Jump on (IX) $\neq$ 0	B-13
73	JOF	Jump on Overflow	B-12
74	JMP	Jump Unconditional	B-12
75	JIX	Jump on Index Register Incremented	B-14
77	NOP	No Operation	B-16

APPENDIX G

ALPHABETICAL INSTRUCTION LIST

Mnemonic	Code	Name	Page
ADD	10	Add (EA) to (A)	B-5
ADM	20	Add Magnitude of (EA) to (A)	B-6
ADX	54	Add to Index Register	B-12
ALS	41	A Register Left Shift	B-8
ANA	15	Logical AND to A	B-8
ARS	40	A Register Right Shift	B-8
*BCD	36	BCD to Binary Conversion	B-7
*BIN	37	Binary to BCD Conversion	B-7
CRA	60	Clear A	B-5
DIV	35	Divide	B-7
DMB	32	Dump Memory Block	B-14
ERA	17	Logical Exclusive OR to A	B-8
ENBI	51	Enable Interrupt	B-15
FMB	31	Fill Memory Block	B-14
HLT	00	Halt	B-16
IAB	57	Interchange (A) and (B)	B-5
INA	52	Input to A	B-15
INAM	52	Input to A Masked	B-15
INHI	51	Inhibit Interrupt	B-15
INM	07	Input to Memory	B-14
IRX	67	Increment, Replace, and Load Index Register	B-13
ITC	51	Interrupt Control	B-15
JIX	72	Jump on (IX) $\neq$ 0	B-13
JMP	74	Jump Unconditional	B-12
JOF	73	Jump on Overflow	B-12
JPL	70	Jump on (A) Plus	B-11
JRT	25	Jump Return	B-11
JST	27	Jump Store	B-11
JIX	75	Jump on Index Register Incremented	B-14
JZE	71	Jump on (A) Zero	B-12
LDA	24	Load A with (EA)	B-5
LDB	23	Load B with (EA)	B-5
LDX	56	Load Index Register	B-13

* Denotes optional instruction

Mnemonic	Code	Name	Page
LGL	47	Logical Left Shift A Register	B-10
LLR	43	Long Left Rotate A and B Registers	B-9
LLS	45	Long Left Shift A and B Registers	B-9
LRR	42	Long Right Rotate A and B Registers	B-9
LRS	44	Long Right Shift A and B Registers	B-9
MPY	34	Multiply	B-6
NOP	77	No Operation	B-16
NRM	46	Normalize Shift A and B Registers	B-10
OCP	53	Output Control Pulse	B-15
ORA	16	Logical OR to A	B-8
OTA	50	Output from A	B-15
OTAM	50	Output from A Masked	B-15
OTM	22	Output from Memory	B-14
RND	62	Round (A)	B-7
SBM	21	Subtract Magnitude of (EA) from (A)	B-6
SCL	65	Scale Left A and B Registers	B-10
SCR	64	Scale Right A and B Registers	B-10
SKG	12	Skip for (A) > (EA)	B-11
SKN	13	Skip for (A) ≠ (EA)	B-11
SKS	61	Skip for Sense Line Not Set	B-15
SMP	30	Step Multiple Precision	B-6
STA	05	Store (A) in (EA)	B-4
STB	03	Store (B) in (EA)	B-4
STC	04	Store Command Portion of (A) in (EA)	B-4
STD	06	Store Address Portion of (A) in (EA)	B-5
STX	66	Store Index Register	B-10
SUB	11	Subtract (EA) from (A)	B-5
TAB	55	Transfer (A) to B	B-5
TAX	63	Transfer Address Portion of A Register to Index Register	B-13
XEC	02	Execute	B-16